Communicating Security Agents

Robert Filman and Ted Linden
Software Technology Center
Lockheed Martin Missiles & Space
3251 Hanover Street O/H1-41 B/254G
Palo Alto, California 94304
{Filman|Linden}@stc.lockheed.com

Abstract

This paper discusses the potential effectiveness of ubiquitous, communicating, dynamically confederating security agents for monitoring and controlling communications among the components of preexisting applications. These agents remember events, communicate with other agents, draw inferences, and plan actions to achieve security goals. Key features of this approach are: (1) linguistic mechanisms for specifying agents, security models, and communications, (2) compilation mechanisms that automatically create and install agents as wrappers around existing application components, (3) algorithmic definitions of how agents communicate to increase the security of systems, and (4) a library of agent code fragments out of which the compilation mechanism builds actual agents. Automating the generation of security agents raises the possibility of the cost-effective generation of enough redundant agents to tolerate some erroneous or subverted elements.

Introduction

The Internet is forcing a paradigm shift in computer security. Rather than the traditional focus on controls that are simple, passive, verifiable, and built-in, open network systems need computer security that is:

- Flexible and context sensitive
- Active in responding to threats
- Reliable through redundant checking
- Incrementally incorporated into existing systems

We believe the first step toward achieving these goals is to wrap conventional software components with programs that analyze communications into and out of applications, monitoring and controlling these communications as appropriate. To effectively perform these tasks, such programs need inherent goals, independent processing, communication facilities, and memory mechanisms. This combination of features defines *software agents* [1, 2]. Building on the notions of robots, softbots (software robots) and safety, we call such agents SafeBots. Individual security agent programs are called safebots.

While there are many challenges facing this paradigm for security, ongoing gains in information technology favor its ultimate success. Key advances include

- **Distributed systems.** Agents can exploit hardware isolation and encrypted communications to reduce single sources of catastrophic failure and implement redundant controls.
- **Decreasing hardware costs.** More processing and communications bandwidth can now be devoted to security without degrading response. Eventually, it will become cost-effective to install safebots on difficult-to-subvert, dedicated processors.
- **High-level protocol standards.** Emerging standards like WWW and CORBA make it increasingly practical to understand component interactions and to automate the generation of component wrappers.
- **Very high level specification languages.** Improving technology for automatically compiling specifications into operational software allows the creation of security systems from high-level requirements.

This paper describes an architecture for secure computing based on SafeBots. We note that we are only at the initial stages of experiments and implementation of the mechanisms described below.

Safebots as wrappers for application components

The modern Internet is composed of many independent domains, sharing a common set of protocols but often radically different security policies. Lacking centralized authority, it follows that security mechanisms based on imposing rules on others are doomed to failure. SafeBots provides an integrated architectural framework for addressing the security policies of heterogeneous information systems. A safebot provides the "locally appropriate" security for a resource, without imposing local constraints on the global system. SafeBots allow the construction of "walls" and "gates" around one's own territory that enable cooperation and commerce with one's neighbors, without having to completely trust those neighbors to be well-behaved.

Safebots monitor communications by wrapping an application's components [3]. In wrapping, a component, application, or computer, X, is replaced by another component, application, or computer, Y, such that Y receives all messages to and from X, censors or edits them, and passes them on to X or an alternative recipient. (Proxy servers are thus a simple form of wrapper.) Safebots can detect errors or suspicious patterns of activities; block inappropriate actions; require further authentication before allowing access; add to the history of the user, session, or component; communicate with other safebots about potential intrusions; fix or randomize the duration of the component call to thwart use of timing covert channels; and check that responses do not leak sensitive information.

Not all safebots are wrappers. Some safebots are *agencies* that serve as repositories of information and behaviors (and are thus a form of *mediators* [4]). Agencies communicate with other safebots. Safebot agencies provide a mechanism for controlled sharing of information about users, computers, sites, system status, normal patterns of behavior, histories of intrusions, recent attack patterns, corrupt software, and other safebot agencies. Some safebot agencies are expert assistants supporting security officers. By being voluntary services with limited trust in other safebots, agencies conform to open networks composed of many independent administrative domains. A given safebot may confederate with different agencies for different purposes.

Figure 1 shows how SafeBots preserves the structure and code of a distributed application while extending it to be a highly secure and survivable application.

Safebot collaboration and communication

We illustrate the value of active, communicating agents with examples of potential agencies. Many of these agencies accumulate information from wrapper agents, manage, summarize, and refine the information, and redistribute it as appropriate to other agents and to security officers.

Agencies that manage information about intrusions

- **Security status agencies.** Safebots wrapping application components can subscribe to security status agencies, informing these agencies of anomalous behaviors and modifying their access mechanisms in response to alert warnings from the security status agencies.

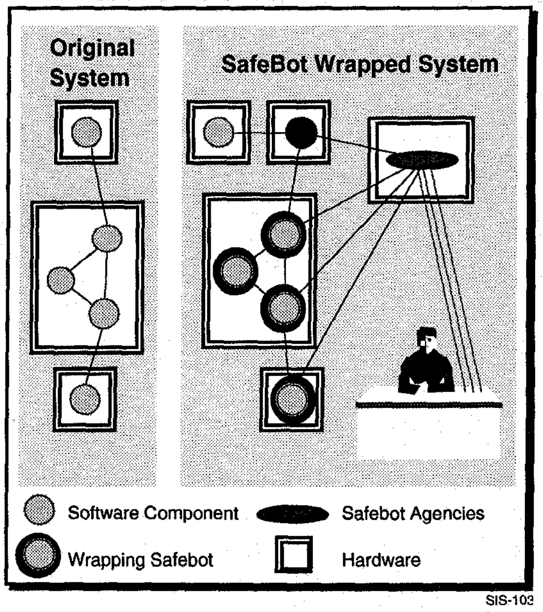

Figure 1: Existing applications evolve into survivable applications by automated addition of safebot wrappers, agencies, and expert assistants

- **Behavior profiles agencies.** These agencies generate a profile of the expected behavior of authorized users with respect to different resources. Safebots protecting a resource use that profile to detect anomalous behaviors. (Forrest *et. al*'s work [5] on characteristic behavior patterns of programs is an example of biologically-inspired data for such agencies; Mukherjee *et. al* [6] discuss novel intrusion detection mechanisms.)
- **Attack patterns agencies.** These agencies manage information about attack methods so the signature of newly detected viruses, scanners, worms, robots, and other attacks can be distributed rapidly to safebots that monitor for such attacks.

Agencies that manage information about users

- **Authentication agencies.** Conventional authentication is usually a rigid, static decision removed from supporting context like the user's location, recent terminal idle time, and the session's recent history of anomalous or suspicious actions. The authentication agency collects reports about user actions and dynamically determines a confidence level in the user's identity. A safebot protecting a critical resource may check with the user's authentication service before granting a request, and then may demand redundant authentication. For example, the authentication service could maintain a list of user-provided memories that no one else is likely to know. The ad-

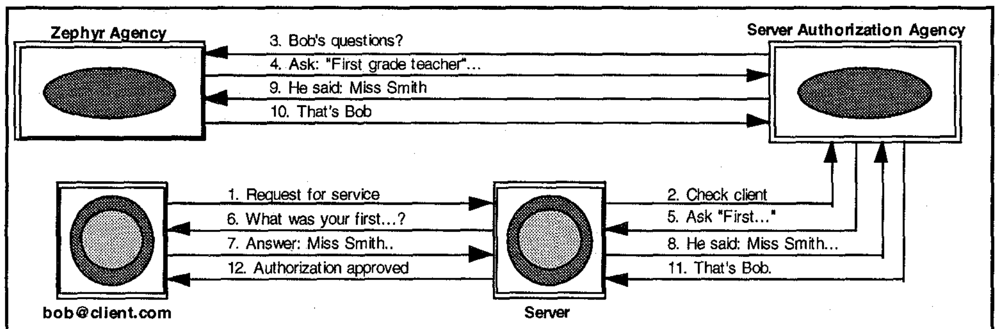

Figure 2: Authentication agencies verify an individual's identity.

vantage of this approach is that it requires no special hardware and can be used when the user is at a location where authentication hardware is missing or broken. Such a protocol is illustrated in Figure 2.

- **Service-worthiness agencies.** Computer security systems traditionally divide potential users into account holders (with privileges), and guests (everyone else, with more limited access). In the networked universe, finer classifications of individuals are needed. For example, just as we won't want to provide any services to an individual who has been recognized elsewhere as a cracker; we may want to provide premium service to an individual recognized as a "big spender." This is realized by having a safebot ask its "service-worthiness agency" about a potential client. Much like a "consumer credit agency," the service-worthiness agency would take experience reports from its customers about individuals and sites and develop profiles of those entities. Clearly, the agents need to respect privacy constraints and plan context-specific responses. Within more structured organizations, the service-worthiness agency will securely identify the access privileges of the user (clearances, kinds of access rights, level of trust) so that administratively remote resources can make access control decisions based on the user's privileges as attested to by a trusted organization.

Agencies about services

- **Service authentication.** Just as services check on users, users check with a service-authenticating agencies before trusting a network service. Certificate authorities (e.g., Verisign [7]) are initial instances of such agencies.
- **Downloading agencies.** Users and their organizations will want to check the trustworthiness of applets before downloading them. Since applets can be loaded with various privileges, many kinds of approval are possible. The downloading agency may provide the applet, present a cryptographic checksum for the applet, or just attest to some degree of trustworthiness for the applet's source.

- **Agencies about safebots.** If safebots completely trusted each other, a rogue or subverted safebot could easily cause catastrophic failure. Safebot agencies attest to the trustworthiness of other safebots. They know which safebots are more susceptible to subversion, collect reports about suspicious acts of safebots, and dynamically adjust their trustworthiness ratings. (Reasoning about network trust is examined by Blaze *et. al* [8] and Röscheisen and Winograd [9].)

Agencies supporting security officers

- **Expert associates.** In some sense, all security agents support the security officer, but some agents can be designed specifically as expert associates to the security officer. These expert associates gather information from other agents, present it to the security officer, and support the security officer in responding to ongoing attacks. A key point is that the security officer can interrogate the safebots about current and recent events and dynamically modify their operating procedures, all as a natural result of the safebots' ability to communicate.

Protecting safebots from bypass and subversion

SafeBots builds on, complements, and extends the security provided by encryption mechanisms. We assume that encryption protects safebot-to-safebot communications from eavesdropping and spoofing. The communications of the application programs being protected may or may not already be encrypted. If they are, safebot wrappers monitor the communications before encryption and after decryption. If the application communications are not encrypted, safebot wrappers are a convenient way to add encryption and safebot agencies are a way to support key management. Encryption can also help sequester application components and prevent the safebot from being bypassed.

Safebots must be protected from subversion of their underlying operating system. Approaches for dealing with this threat include:

- **Running safebots on dedicated hardware.** That way, subversion of an operating system under the application does not subvert its protecting safebots.
- **Continuous mutual vetting by distributed safebots.** An intruder who has subverted an operating system may find it difficult to subvert all the

safebots running on that machine and arrange for them to continue behaving in ways that do not arouse suspicions.

- **Varying safebot trust in other safebots based on the environments in which they run.** Safebots that run in more secure environments with more checking can be trusted more than safebots in more open systems.

Constructing safebots

A key pragmatic problem for SafeBots is the multiplicity of component interfaces and security requirements. The goal of the SafeBots architecture is to reuse common solutions while making it cost effective for agents to handle the specific application protocols and protection requirements of each system. The components of the architecture raise the level of safebot specification and automate safebot construction, making it easier to impose security mechanisms on existing applications, and allowing cost-effective experiments with alternative security policies. The key components of the SafeBots architecture are:

- **OntoSec:** a common language for specifying the security requirements of each application and for supporting communication among safebots
- **Swathe** a compilation and automatic programming system for weaving library components together to build safebots
- **SecLib**: a library of reusable components and fragments for building safebots

Figure 3 illustrates the relationship of these components in producing safebots.

OntoSec

Two linguistic issues arise with SafeBots: (1) describing the specification of safebots to the compiler and (2) the language used by safebots for communication. For generality, we define these as parts of a common language, *OntoSec* (for "ontology for security.") OntoSec represents security requirements, specifications, goals, actions, events, and knowledge of agents. For example, OntoSec can describe protocols; the security properties of resources and components; the privileges of users and sessions; events, actions to be taken on events, and semantic bindings for implementing those actions; and histories of users and systems. OntoSec provides a vocabulary for specifying the security properties to be enforced by safebots and for safebots to communicate with each other and with security personnel.

Important dimensions of OntoSec are that:

- It is expressive enough to specify policies, status, knowledge, beliefs, and concerns. It must support safebots in determining the level of trust to place in messages and requests from other safebots.
- It is directly computable. That is, we want a system that infers the consequences of a collection of security statements in a reasonable amount of time.
- It provides a way of unifying programmatic behavior with reasoning.

We are exploring the use of a formal, logical language for OntoSec, with individuals for the actors of security (e.g., people, sessions, events, histories, applications), primitive relations for properties of these individuals (e.g., permissions, locations, privileges), a semantic attachment mechanism for tying syntax to code, and modal operators for expressing notions such as requirements, beliefs and probability. Ontologies are an important theme in current AI research [10]. Examples of ontological approaches to security include Yialelis *et. al* [11] and the deontic logic work of Bieber and Cuppens [12]. An important element in the generation of communicating intelligent agents is an appropriate underlying communication protocol; KQML [13] is one such language.

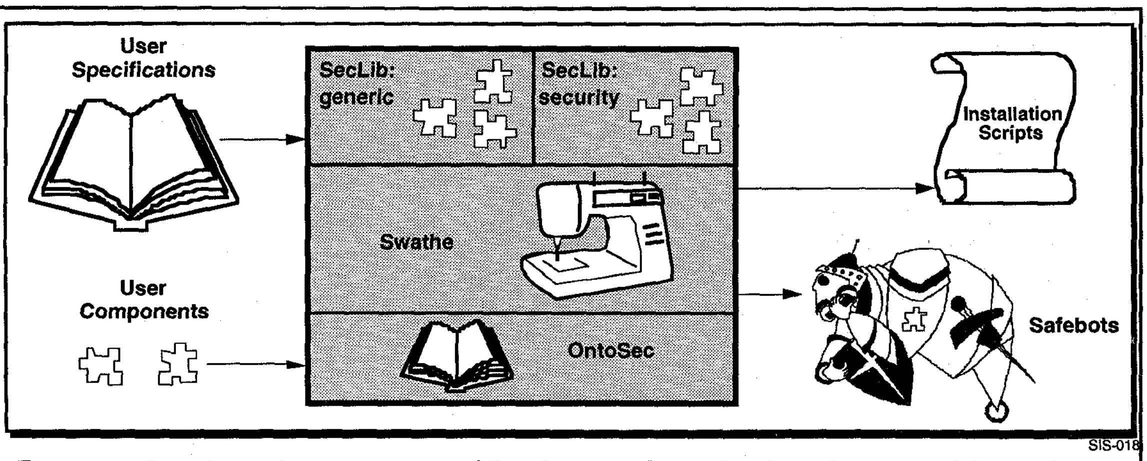

Figure 3: **Swathe takes user specifications written in OntoSec, combines them with the library of security components and produces safebots and installation scripts for these safebots.**

Swathe

SafeBots is based on the wrapability of applications and components. We assume that components to be protected (1) are *specified*—that is, have a well-defined, formally representable interface, (2) can be *sequestered*—that is, placed where intruders cannot invoke them directly, and (3) can be *substituted*—that is, a replacement component can be introduced into the system in their place. This replacement component supports the specified interface, performs whatever security actions are associated with the call, and invokes the sequestered, original function to do the actual work. Examples of wrapped components range from network proxy servers through UNIX executable shells on to tracing in Lisp.

Manual wrapping is labor-intensive, cumbersome, error-prone, and inconsistent. We propose a tool for automatically creating safebots, Swathe. Swathe takes as inputs:

- The interface definitions of the application components
- OntoSec specifications of desired security properties
- A library of security algorithms and safebot code fragments (SecLib)
- The physical organization of the system (e.g., locations of existing applications)

Swathe constructs a wrapped application or component that conforms to the specified security properties and a script for installing the wrapper [14].

Note that Swathe is not dealing with the semantics of application component interfaces—security programmers write SecLib routines that can do things with the information content being passed. Rather, the automatic programming of Swathe adjusts the safebots code to deal with the syntax of communications—a more tractable and quite useful activity.

An important element of this scheme is the existence of SecLib—Swathe works primarily by selecting appropriate elements from this library and coherently knitting them together.

When a safebot intercepts a method invocation to or from the wrapped component, Swathe makes additional parameters available to the safebot. These parameters identify the calling session, its security context, and the responsible human source of the call. (Additional parameters of this ilk are already passed by some CORBA implementations of remote procedure calls, e.g. Orbix™; the magic cookie protocol of HTML is an another contextual mechanism that could be exploited).

SecLib

SecLib is an expanding collection of algorithms, mechanisms, and safebot code fragments that understand OntoSec and can be automatically assembled into safebots. These fragments enable safebots to:

- Sense and evaluate their environment to detect security threats
- Understand and reason about OntoSec specifications
- Communicate with other safebots
- Reason about actions to best enforce security policy in the current context
- Reason about communications received from other safebots. (For example: Have they been subverted? What information should I send them? Should we collectively ostracize them? How does their communication affect my understanding of my context?).

Additional safebot fragments implement specific security algorithms. They focus on redundant user authentication, data aggregation, statistical analysis, access control, denial of service due to system overloading, and other specific threats and controls.

Swathe weaves these safebot fragments into safebots capable of enforcing OntoSec specifications for the application component around which they are wrapped. One potentially radical approach to security is to include multiple versions of algorithms and fragments in SecLib, allowing Swathe to introduce "genetic diversity" into its space of wrapped organisms. Thus, a hole in one particular implementation of a component does not render vulnerable every user of the semantics of that component. This could be used to explore the hypothesis that a diverse ecology of security mechanisms is less vulnerable to a single-disease catastrophic failure than a monoculture of identical organisms. Similarly, a system composed of security elements that trust each other "less than completely" and whose "genetic code" varies is less exposed to a single point of failure, much as a biological systems use multiple immune responses to protect against a variety of parasites.

Safebots dynamically form federations, joined by interest in the behavior of particular users, systems, or sessions [3]. They check on each other and evaluate the trust they place in communications from other safebots. Since safebots are created by the owners (or "partial owners") of components, SafeBots technology supports the realization of systems embodying multiple, overlapping administrative concerns.

Limitations of agent-based approaches

While safebots offer many advantages over simple, static architectures for security, they also have several disadvantages, especially in the near term:

- Initially, safebots will make security administration more complex. It will be difficult for the average security officer to understand everything that is going on. Configuring safebots to check on each other will be complex. Eventually, the benefits of redundancy, high-level specifications, and visualization will make the security officer's job easier, but it will be some time before we achieve enough redundancy to cover mistakes in administering security.

- Safebots wrap only application components that have well-defined application program interfaces (APIs), specified in a supported interface definition language (IDL). Applications with complex GUI's or interpreters (e.g., shells, programming environments) are not good candidates for wrapping.
- Safebots themselves can become a source of catastrophic failure. Subverting a safebot could become a way to attack systems, and inept security designers could design safebots that reduce rather than enhance survivability. SafeBots is designed so safebots can check on each other and limit their trust in other safebots. We need to determine the extent to which these mechanisms are practical.
- Safebots can degrade the performance and response time of a system. At a time of crisis, heightened safebot activity could tie up a system just when it is most needed. Eventually, safebots will reason about the effect they are having on performance. Faster hardware and careful design are keys to long-range mitigation of this concern.

Conclusion

SafeBots is a vehicle for experiments with cost-effectiveness of redundant security agents distributed pervasively throughout applications. The long term goal of SafeBots is to make defensive controls dramatically less expensive and force intruders to breach redundant barriers—turning the advantage to security defense and fundamentally changing the balance between penetraters and security personnel.

Acknowledgments

Our thanks to Todd Heberlein and Karl Levitt for discussions of the ideas in this paper.

References

[1] M. J. Wooldridge and N. R. Jennings, "Agent Theories, Architectures, and Languages: A Survey," in M. J. Wooldridge and N. R. Jennings (Eds.) Proc. ECAI-94, Workshop on Agent Theories, Architectures and Languages, Springer-Verlag Lecture Notes in Artificial Intelligence-890, 1995, pp. 1–39.

[2] D. Riecken, Guest Editor, "Special Issue: Intelligent Agents," *CACM 37*, 7, 1994.

[3] M. Genesereth and S. P. Ketchpel, "Software Agents," *CACM 37*, 7, 1994, pp. 48–53.

[4] G. Wiederhold, "Mediators in the Architecture of Future Information Systems," *IEEE Computer 25*, 3, 1992, pp. 38–49.

[5] S. Forrest, S. A. Hoffmeyr, A. Somayaji, and T. A. Longstaff, "A Sense of Self for Unix Processes," *Proc. IEEE Symposium on Security and Privacy,* 1996, pp. 120–128.

[6] B. Mukherjee, L. T. Heberlein, and K. N. Levitt, "Network Intrusion Detection," *IEEE Network 8*, 3, 1994, pp. 26–41.

[7] Verisign. http://www.verisign.com.

[8] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized Trust Management, *Proc. IEEE Symposium on Security and Privacy*, 1996, pp 164–173.

[9] M. Röscheisen and T. Winograd, "A Communication Agreement Framework for Access/Action Control," *Proc. IEEE Symposium on Security and Privacy*, 1996, pp 154–163.

[10] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, & W. R. Swartout. "Enabling Technology for Knowledge Sharing," *AI Magazine, 12,* 3, 1991, pp. 16–36.

[11] N. Yialelis, E. Lupu, and M. Sloman, "Role-Based Security for Distributed Object Systems", *Proc. IEEE Fifth Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises,* 1996.

[12] P. Bieber and F. Cuppens, "Expression of Confidentiality Policies with Deontic Logic," in J. Meyer and R. Wieringa (Eds.) *Deontic Logic in Computer Science: Normative System Specification,* Wiley, 1993.

[13] T. Finin, R. Fritzson, D. McKay and R. McEntire, "KQML as an Agent Communication Language," *Proc. 3rd Int'l Conference on Information and Knowledge Management,* 1994.

[14] H. Graves, "Lockheed Environment for Automatic Programming," *IEEE Expert*, Dec. 1992, pp. 15–25.